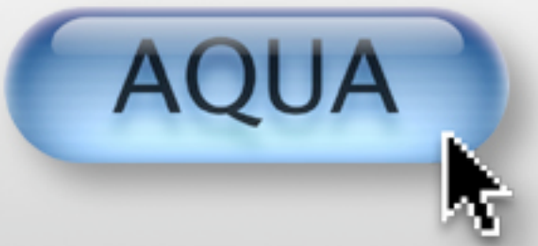
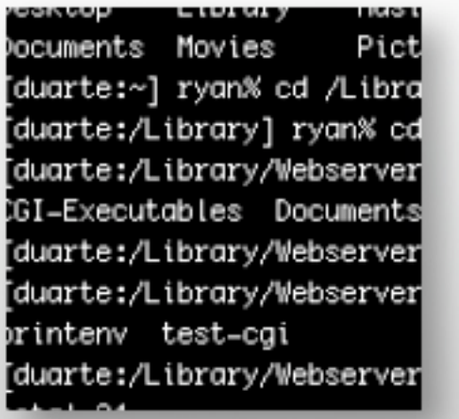
Making a Modern Carbon Application

Using C and C++

Session 423





Making a Modern Carbon Application

Using C and C++

Xavier Legros
Mac OS X Evangelist
xavier@apple.com

Making a Modern Carbon Application

Using C and C++

David McLeod
HIToolbox Engineer

Introduction

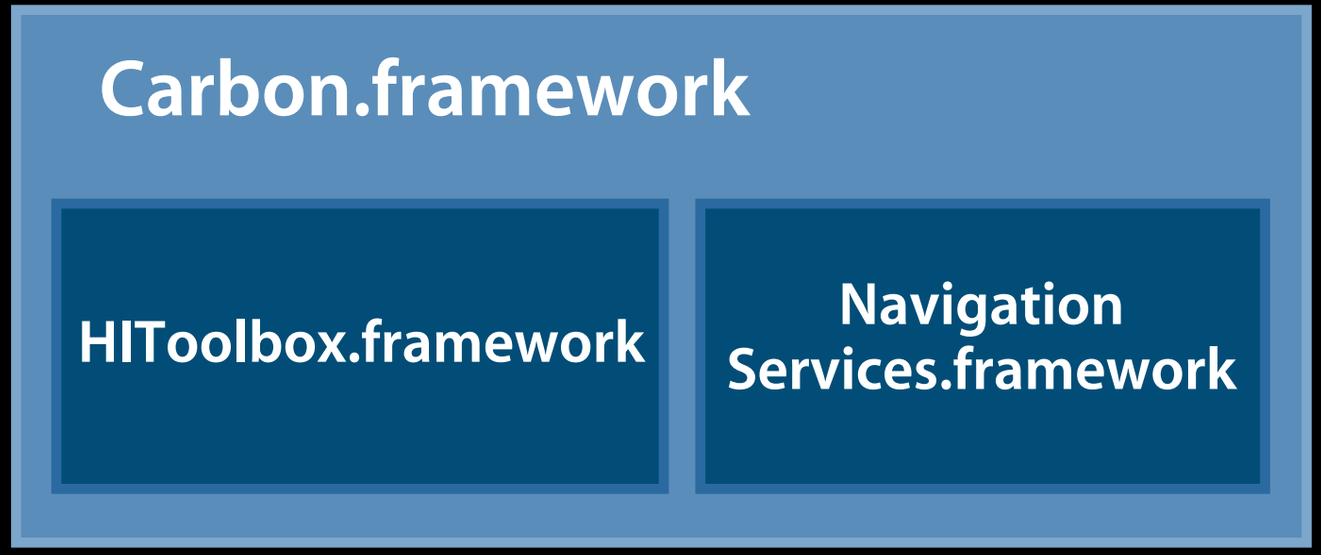
- Who should be here?
- What is a modern Carbon application?
  - Simplified, efficient event dispatching
    - Nib-defined user interface
  - Uses up-to-date OS Services
    - Navigation Services
    - CFPreferences
    - Icon Services
    - Unicode
    - Etc.

What You Will Learn

- How to
  - Make a new Carbon application with XCode
  - Make a user interface with Interface Builder
  - Use interface elements available in HIToolbox
  - Handle events and commands
  - Add Sheets and Standard Alerts in your application
  - Use Navigation Services in your application

Technology Frameworks

- Carbon.framework
  - HIToolbox.framework
  - NavigationServices.framework



What Is Carbon?

- Carbon is the set of APIs used to write C and C++ applications for Mac OS X
- Allow access to Mac OS X services
  - User Interface
  - File System
  - Etc.
- HIToolbox is the Carbon framework used to make application User Interfaces with C and C++

Being a Modern Carbon Application

- Use Xcode
  - Make a mach-o application
  - Application is made as a Bundle
- Use Interface Builder to create your user interface
  - User interface stored in Nib files
- Use current Mac OS X technologies
  - Quartz, Navigation Services, HIViews, etc.
- Use the Carbon Event dispatcher

Being a Modern Carbon Application

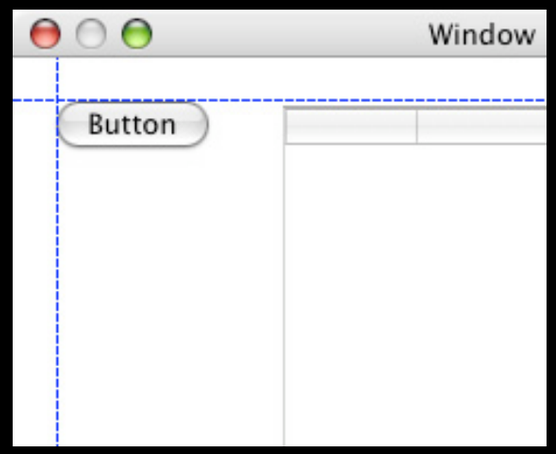
Old Way	Modern
Pascal Strings	CFStrings (Unicode)
WaitNextEvent	CarbonEvents
Standard File	Navigation Services
Single Application File	Application Bundle
Resource Based UI	Nib Based UI
Controls	HIViews
Quickdraw	Quartz

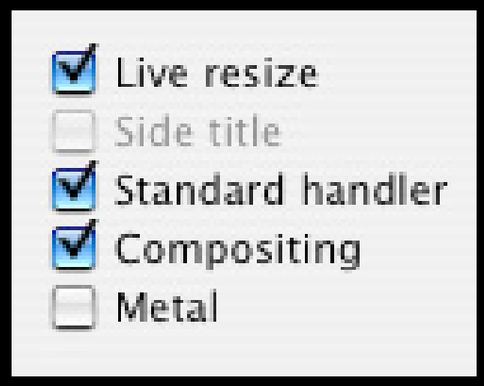
Making an Application With Xcode

Guy Fullerton

A Modern User Interface

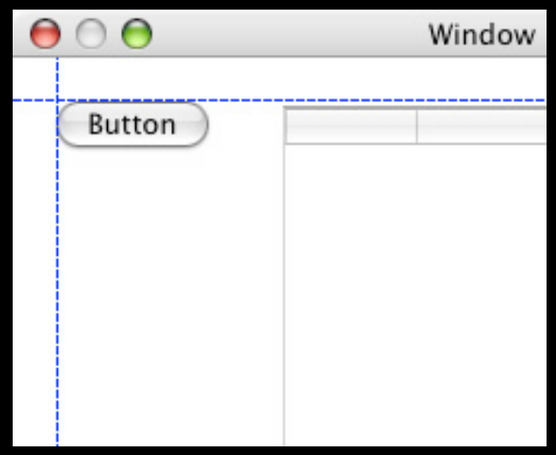
- Adhere to Human Interface Guidelines
  - Interface Builder guidelines feature
  - Use HIToolbox controls
- Use modern, compositing windows
  - Speed benefits
  - It's just a check box in Interface Builder!
- Turn on live resizing
  - It's just a check box in Interface Builder!
- Use the Standard Event Handler
  - Guess where the check box is!

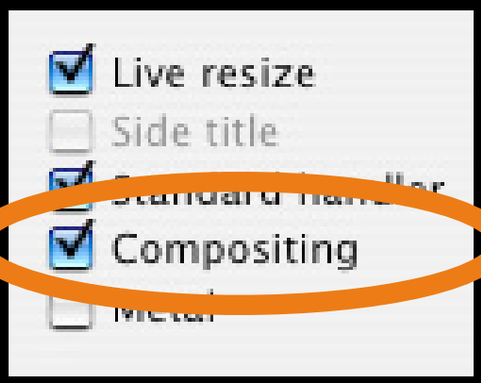




A Modern User Interface

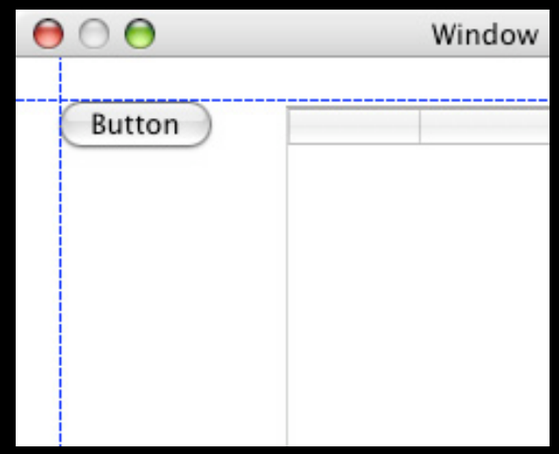
- Adhere to Human Interface Guidelines
  - Interface Builder guidelines feature
  - Use HIToolbox controls
- Use modern, compositing windows
  - Speed benefits
  - It's just a check box in Interface Builder!
- Turn on live resizing
  - It's just a check box in Interface Builder!
- Use the Standard Event Handler
  - Guess where the check box is!

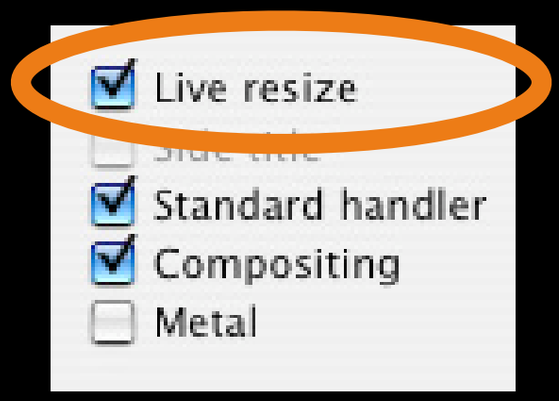




A Modern User Interface

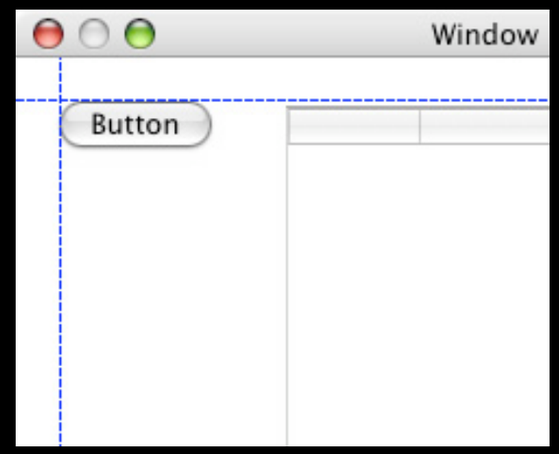
- Adhere to Human Interface Guidelines
  - Interface Builder guidelines feature
  - Use HIToolbox controls
- Use modern, compositing windows
  - Speed benefits
  - It's just a check box in Interface Builder!
- Turn on live resizing
  - It's just a check box in Interface Builder!
- Use the Standard Event Handler
  - Guess where the check box is!

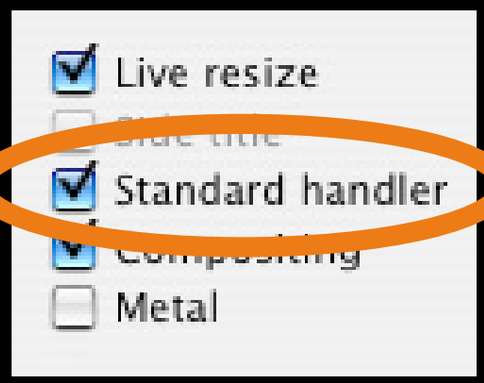




A Modern User Interface

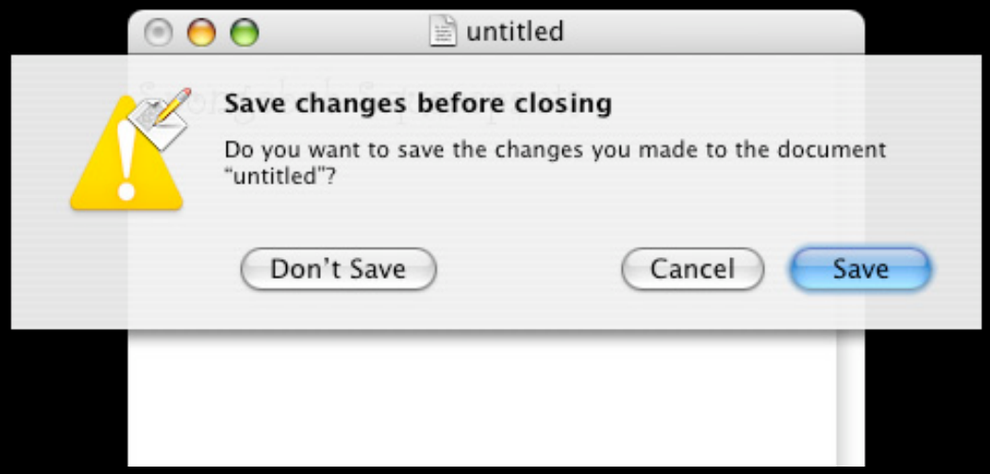
- Adhere to Human Interface Guidelines
  - Interface Builder guidelines feature
  - Use HIToolbox controls
- Use modern, compositing windows
  - Speed benefits
  - It's just a check box in Interface Builder!
- Turn on live resizing
  - It's just a check box in Interface Builder!
- Use the Standard Event Handler
  - Guess where the check box is!

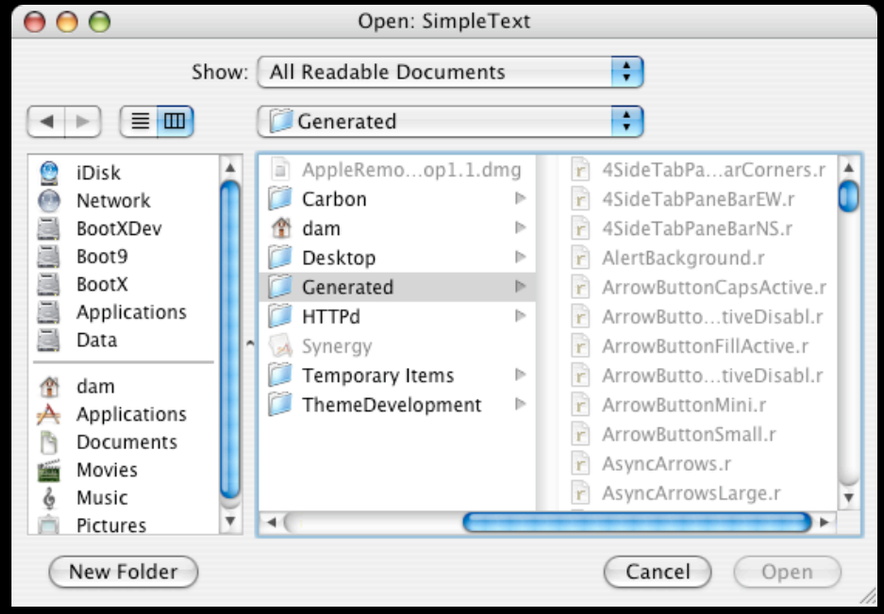




A Modern User Interface, Part 2

- Use sheets or standard alerts where appropriate
- Use Navigation Services to open, save or select files and folders
  - Does the file system work for you
  - Is very customizable



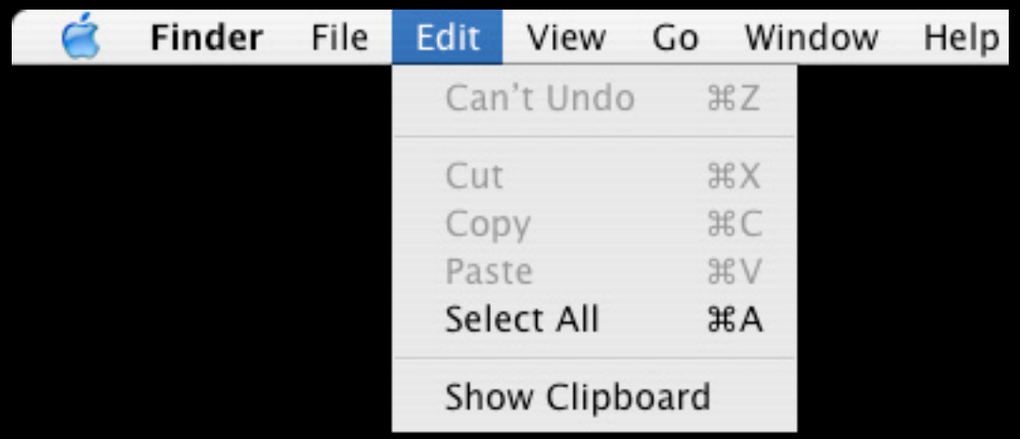


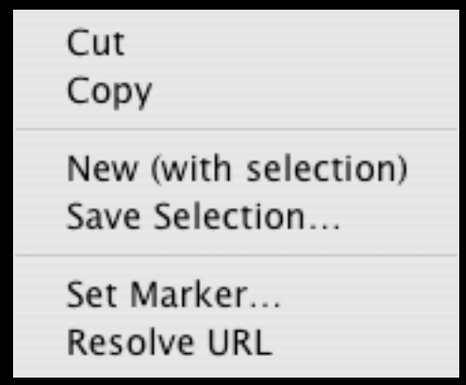
Available UI Elements

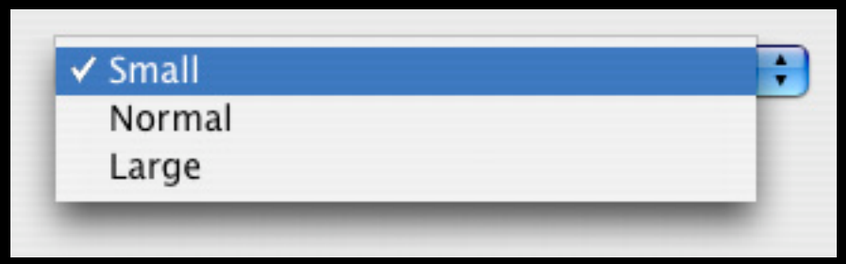
- Menus
- Windows
- Toolbars
- Controls

Available UI Elements

- Menus
- Windows
- Toolbars
- Controls

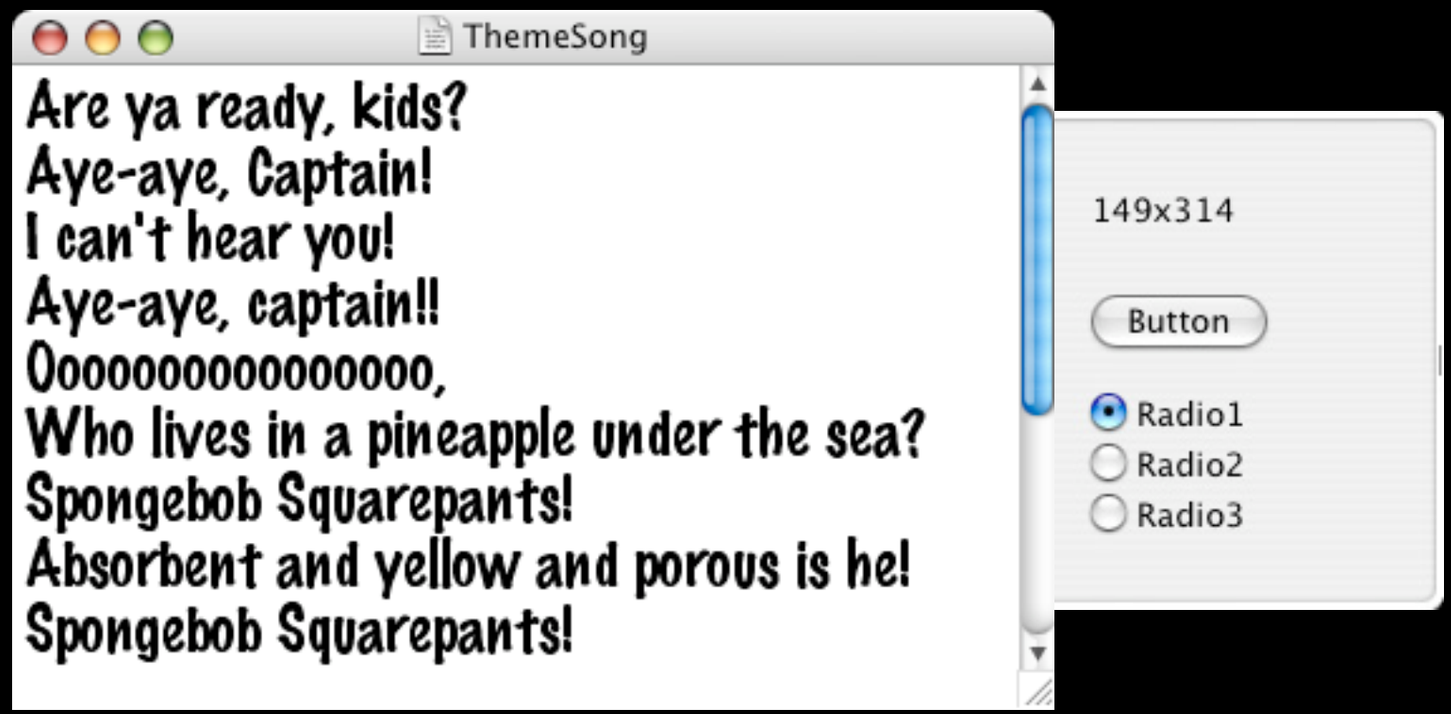


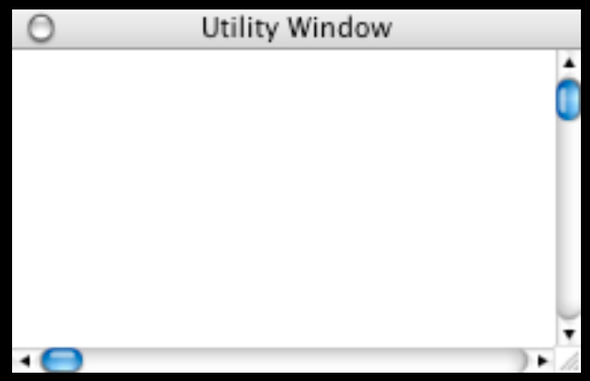


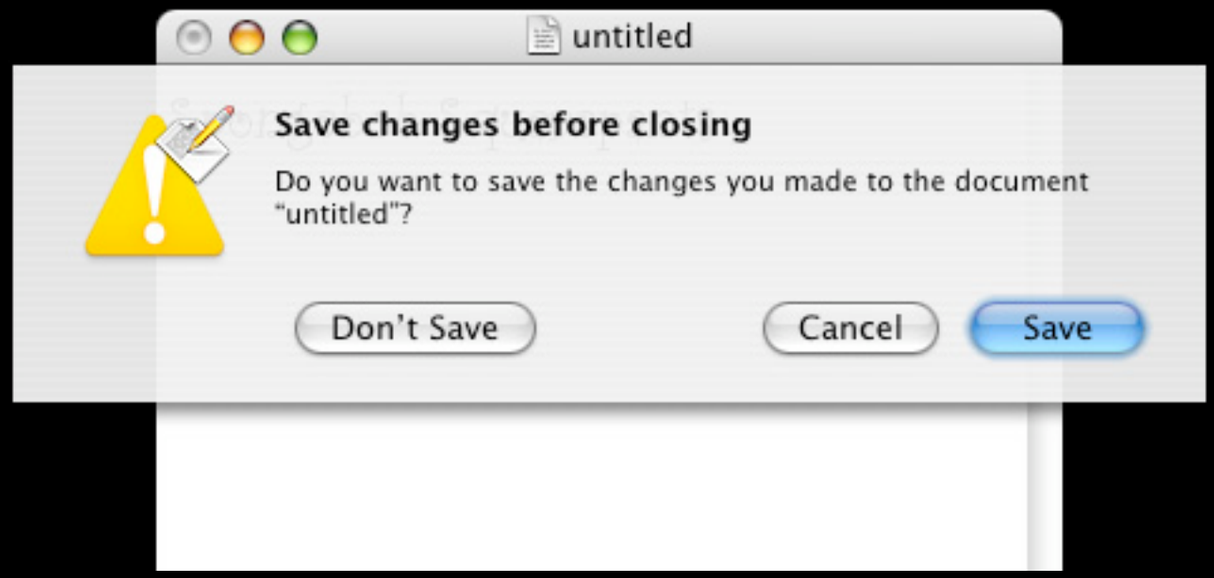


Available UI Elements

- Menus
- Windows
- Toolbars
- Controls

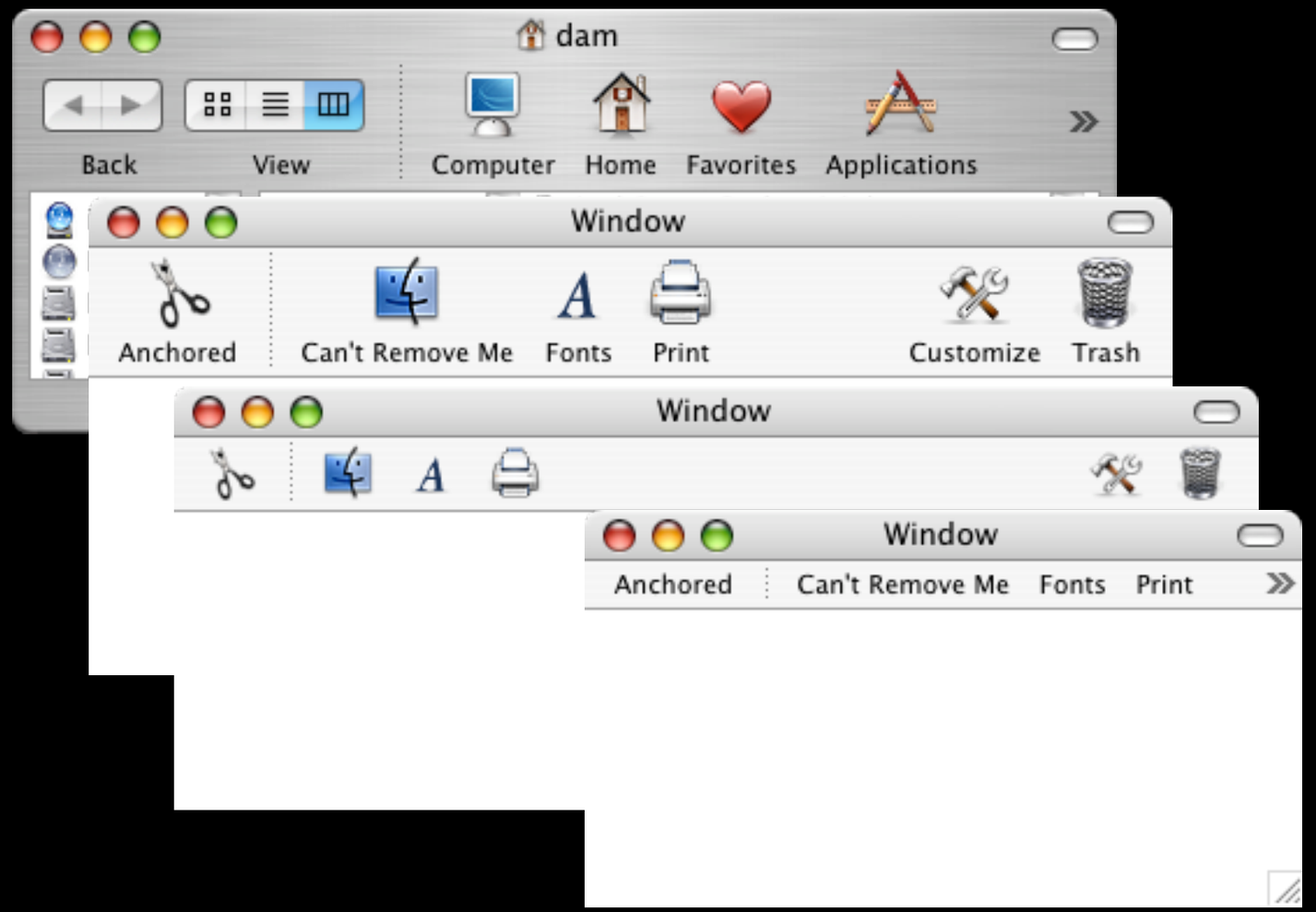






Available UI Elements

- Menus
- Windows
- Toolbars
- Controls

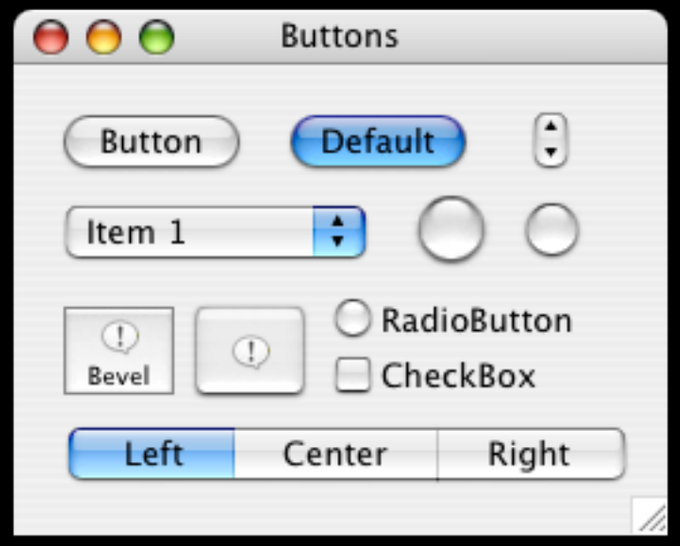


Available UI Elements

- Menus
- Windows
- Toolbars
- Controls
  - Buttons: push button, radio button, check box, bevel button, popup button, little arrows, segment view*
  - Text: edit text field, text view, static text, combo box, clock control, search field
  - Tracks: scroll bar, slider, progress indicator, relevance bar
  - Grouping: tab control, group box, radio group
  - Specialized: chasing arrows, separator bar, image view, image well, icon control, scrolling view, data browser

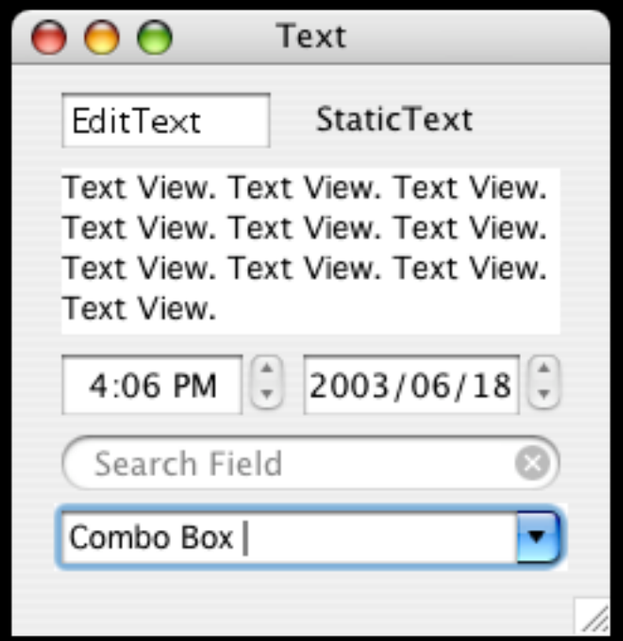
Available UI Elements

- Controls
  - Buttons
    - Push button
    - Radio button
    - Check box
    - Bevel button
    - Popup button
    - Little arrows
    - Segment view*



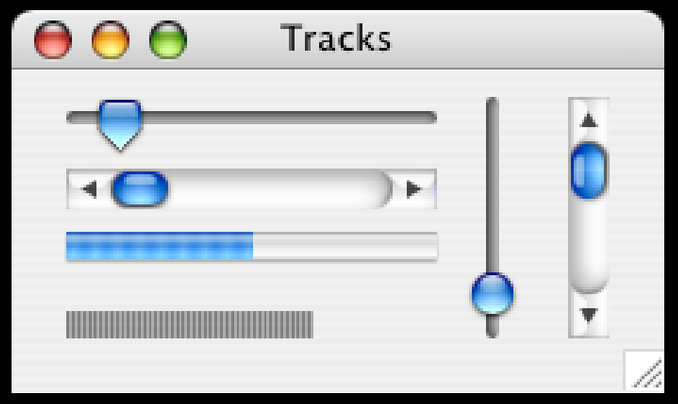
Available UI Elements

- Controls
  - Text
    - Edit text field
    - Static text
    - Text view
    - Clock control
    - Search field
    - Combo box



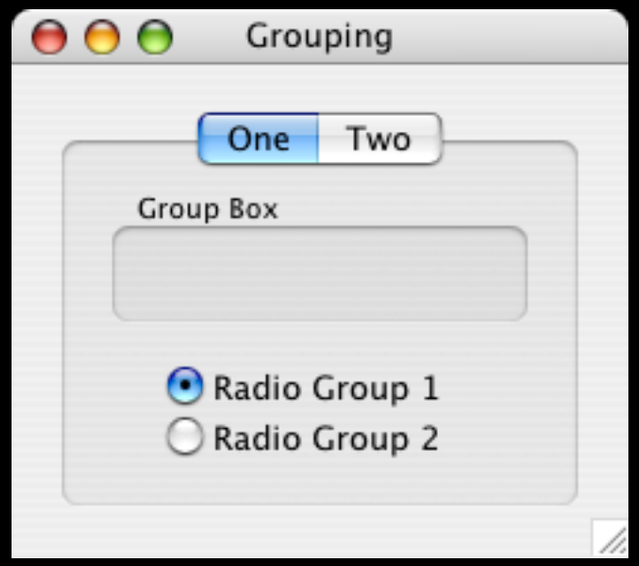
Available UI Elements

- Controls
  - Tracks
    - Scroll bar
    - Slider
    - Progress indicator
    - Relevance bar



Available UI Elements

- Controls
  - Grouping
    - Tab control
    - Group box
    - Radio group



Available UI Elements

- Controls
  - Specialized
    - Image view
    - Chasing arrows
    - Separator bar
    - Icon control
    - Image well
    - Scrolling view
    - Data browser

Available UI Elements

- Menus
- Windows
- Toolbars
- Controls
  - Buttons: push button, radio button, check box, bevel button, popup button, little arrows, segment view*
  - Text: edit text field, text view, static text, combo box, clock control, search field
  - Tracks: scroll bar, slider, progress indicator, relevance bar
  - Grouping: tab control, group box, radio group
  - Specialized: chasing arrows, separator bar, image view, image well, icon control, scrolling view, data browser

Nibs

Loading a Nib

```
OSStatus CreateNibReference(
   CFStringRef         inNibName,
   IBNibRef*           outNibRef );

void DisposeNibReference(
   IBNibRef            inNibRef );
```

Loading Stuff From a Nib

```
OSStatus SetMenuBarFromNib(
   IBNibRef            inNibRef,
   CFStringRef         inName );

OSStatus CreateMenuFromNib(
   IBNibRef            inNibRef,
   CFStringRef         inName,
   MenuRef*            outMenuRef );

OSStatus CreateWindowFromNib(
   IBNibRef            inNibRef,
   CFStringRef         inName,
   WindowRef*          outWindow );
```

Loading Stuff From a Nib

```
OSStatus SetMenuBarFromNib(
   IBNibRef            inNibRef,
   CFStringRef         inName );

OSStatus CreateMenuFromNib(
   IBNibRef            inNibRef,
   CFStringRef         inName,
   MenuRef*            outMenuRef );

OSStatus CreateWindowFromNib(
   IBNibRef            inNibRef,
   CFStringRef         inName,
   WindowRef*          outWindow );
```

Loading Stuff From a Nib

```
OSStatus SetMenuBarFromNib(
   IBNibRef            inNibRef,
   CFStringRef         inName );

OSStatus CreateMenuFromNib(
   IBNibRef            inNibRef,
   CFStringRef         inName,
   MenuRef*            outMenuRef );

OSStatus CreateWindowFromNib(
   IBNibRef            inNibRef,
   CFStringRef         inName,
   WindowRef*          outWindow );
```

RunApplicationEventLoop

- Starts event dispatcher
- Only returns when the application quits
  - does standard event handling
  - calls event handlers
- Super extra ultra mega easy to use

```
void RunApplicationEventLoop();
```

Nib Sample—It's a Full App!

```
NibRef                  myNib;
WindowRef               myWindow;

err = CreateNibReference( CFSTR( "MainNib" ), &myNib );

err = SetMenuBarFromNib( myNib, CFSTR( "MyMenuBar" ) );

err = CreateWindowFromNib( myNib, CFSTR( "MyWindow" ),
     &myWindow );
ShowWindow( myWindow );

DisposeNibReference( myNib );

RunApplicationEventLoop();
```

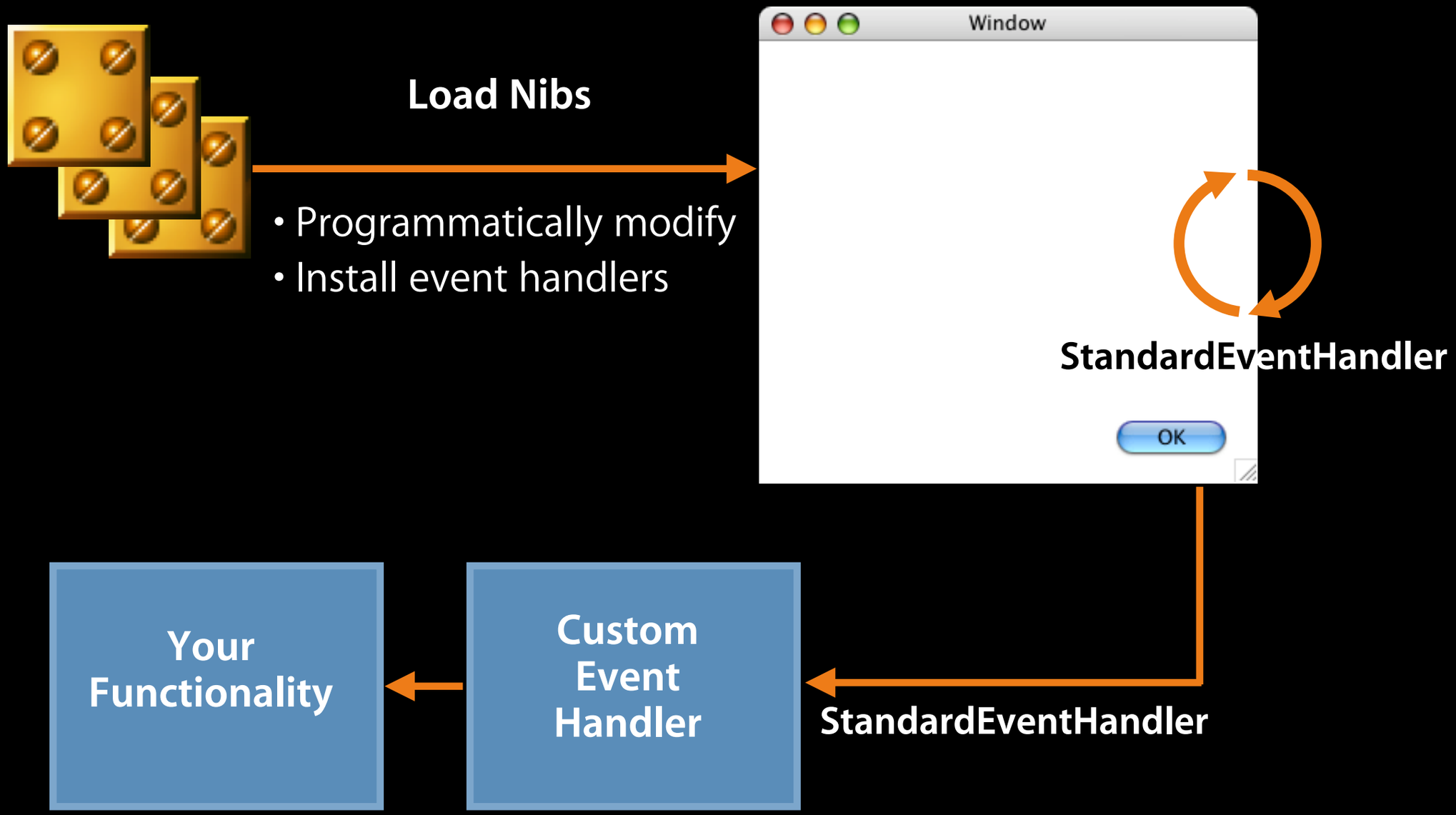
Making a User Interface With Interface Builder

Gi

Architecture of an Application

- Nibs that define the user interface
- Handling of Carbon Events
  - Only listen to what is important to the application
- Handling of HICommands
- Reactive user interface
  - Enabling/Disabling controls, menus and commands
- Other than that, just your application's functionality

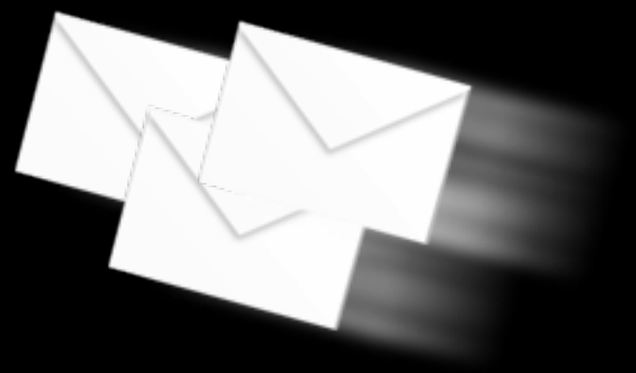
Architecture of an Application



What Are Events?

- Events are the messaging system of HIToolbox
  - Clicks
  - Typing
  - Custom messaging

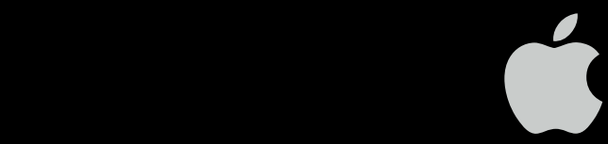
- HIToolbox sends tons of messages!
  - You don't need to handle too many
  - Only handle the ones you care about
  - Ignore most of them!



Standard Event Handling

Standard Event Handling

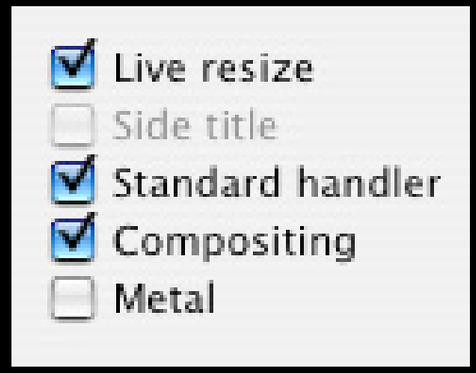
- HIToolbox handles most events automatically
  - Application activation/deactivation
  - Menu tracking
  - Control tracking and animation
  - Window widget handling
  - Window resizing, activation, deactivation, zooming
- The more you use HIToolbox automatic behavior, the better off you are going forward.
- Simplifies your UI code



The StandardEventHandler

- Special event handler for windows
- Handles standard window behavior for free!
  - Moving
  - Resizing, collapsing and zooming
  - Toolbar toggling
- Routes mouse and keyboard events to control
- Check the check box in IB and forget about it!

FREE!



Event Handlers

- Event Classes and Kinds
  - Read CarbonEvents.h
- Event Types
  - Methods, Notifications, Hooks
- Event Targets
  - Application, Windows, Controls, etc.

```
OSStatus MyControlHandler(
    EventHandlerCallRef    inCallRef,
    EventRef               inEvent,
    void*                  inUserData );
```

Example Event Handler

```
OSStatus MyControlHandler(
    EventHandlerCallRef    inCallRef,
    EventRef               inEvent,
    void*                  inUserData )
{
    OSStatus               err;
    ControlRef             control = (ControlRef) inUserData;
    HIRect                 newBounds;

    err = GetEventParameter( inEvent, kEventParamCurrentBounds,
      typeHIRect, 0, sizeof(HIRect), 0, &boundsRect );

    // ... do something with newBounds ...

    return noErr;
}
```

Example Event Handler

```
OSStatus MyControlHandler(
    EventHandlerCallRef    inCallRef,
    EventRef               inEvent,
    void*                  inUserData )
{
    OSStatus               err;
    ControlRef             control = (ControlRef) inUserData;
    HIRect                 newBounds;

    err = GetEventParameter( inEvent, kEventParamCurrentBounds,
      typeHIRect, 0, sizeof(HIRect), 0, &boundsRect );

    // ... do something with newBounds ...

    return noErr;
}
```

Example Event Handler

```
OSStatus MyControlHandler(
    EventHandlerCallRef    inCallRef,
    EventRef               inEvent,
    void*                  inUserData )
{
    OSStatus               err;
    ControlRef             control = (ControlRef) inUserData;
    HIRect                 newBounds;

    err = GetEventParameter( inEvent, kEventParamCurrentBounds,
      typeHIRect, 0, sizeof(HIRect), 0, &boundsRect );

    // ... do something with newBounds ...

    return noErr;
}
```

Installing an Event Handler

```
OSStatus InstallEventHandler(
    EventTargetRef          inTarget,
    EventHandlerUPP         inHandler,
    UInt32                  inNumTypes,
    const EventTypeSpec*    inList,
    void*                   inUserData,
    EventHandlerRef*        outRef );    /* can be NULL */


Shortcuts:

InstallControlEventHandler
InstallWindowEventHandler
InstallApplicationEventHandler
```

Installing an Event Handler

```
OSStatus InstallEventHandler(
    EventTargetRef          inTarget,
    EventHandlerUPP         inHandler,
    UInt32                  inNumTypes,
    const EventTypeSpec*    inList,
    void*                   inUserData,
    EventHandlerRef*        outRef );    /* can be NULL */


Shortcuts:

InstallControlEventHandler
InstallWindowEventHandler
InstallApplicationEventHandler
```

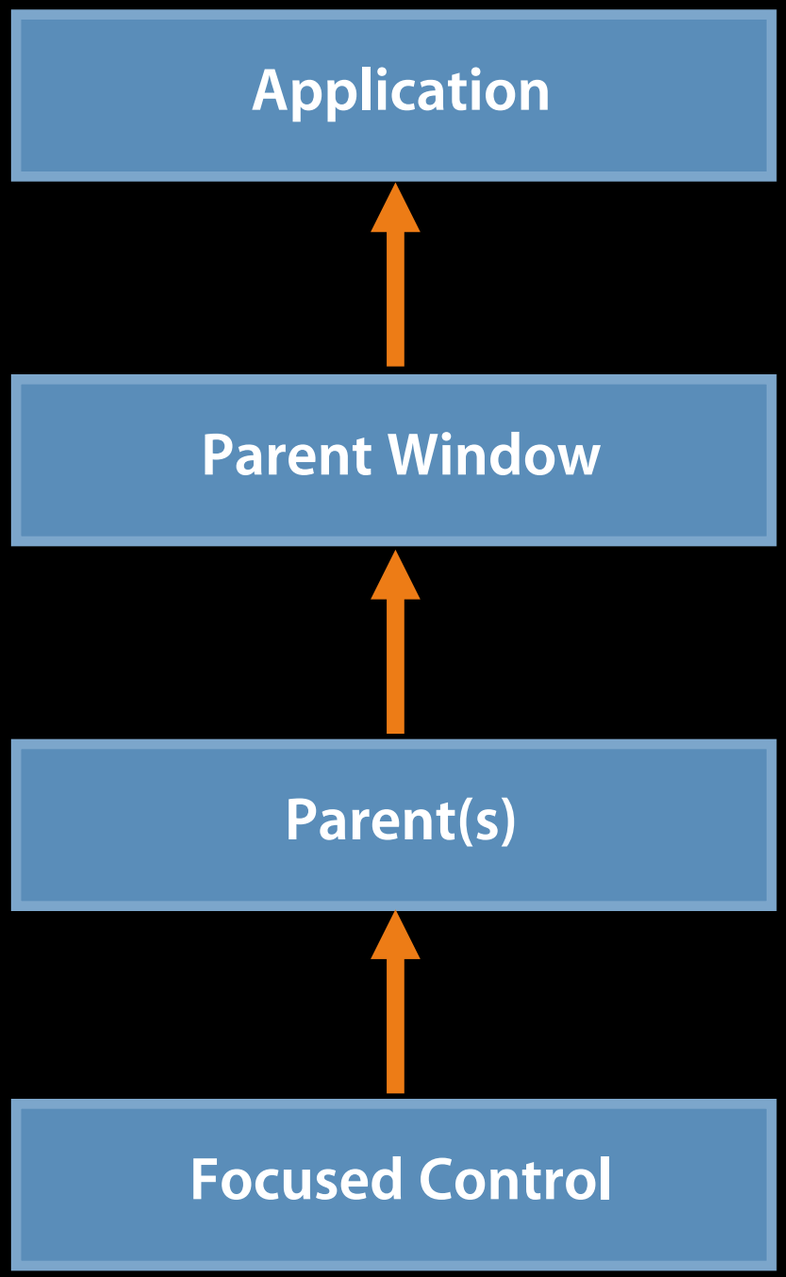
Installing an Event Handler

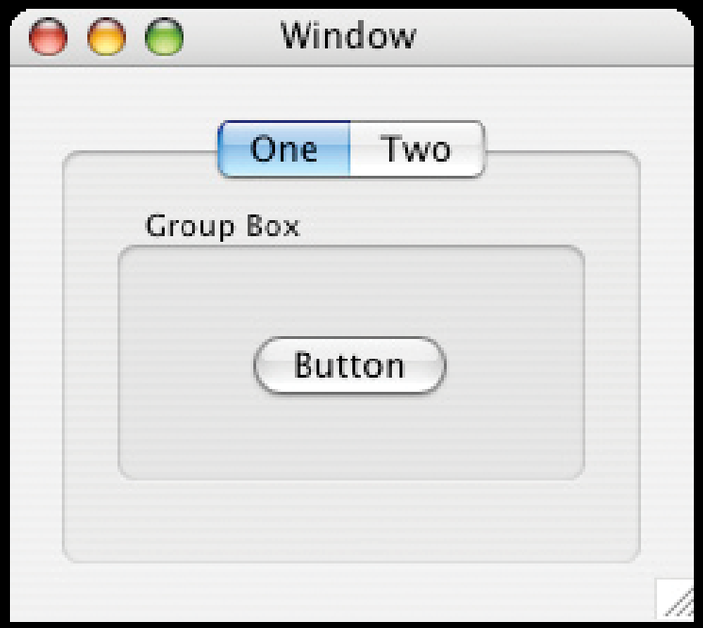
```
// Prototype
OSStatus MyControlHandler(
  EventHandlerCallRef inCallRef,
  EventRef            inEvent,
  void*               inUserData );

const EventTypeSpec   kEventList[] = {
                       { kEventClassControl, kEventControlDraw },
                         ... };

err =  InstallControlEventHandler( myControl,
  MyControlEventHandler, GetEventTypeCount( kEventList ),
  kEventList, myControl, NULL );
```

Example Event Handler

```
OSStatus MyEventHandler(
    EventHandlerCallRef    inCallRef,
    EventRef               inEvent,
    void*                  inUserData )
{
    ...
    switch ( GetEventClass( inEvent ) )
    {
        case kEventClassControl:
            {
                switch ( GetEventKind( inEvent ) )
                {
                    case kEventControlBoundsChanged:
                }
            }
            break;
        case...
    }
    ...
}
```

How Events and Commands Flow





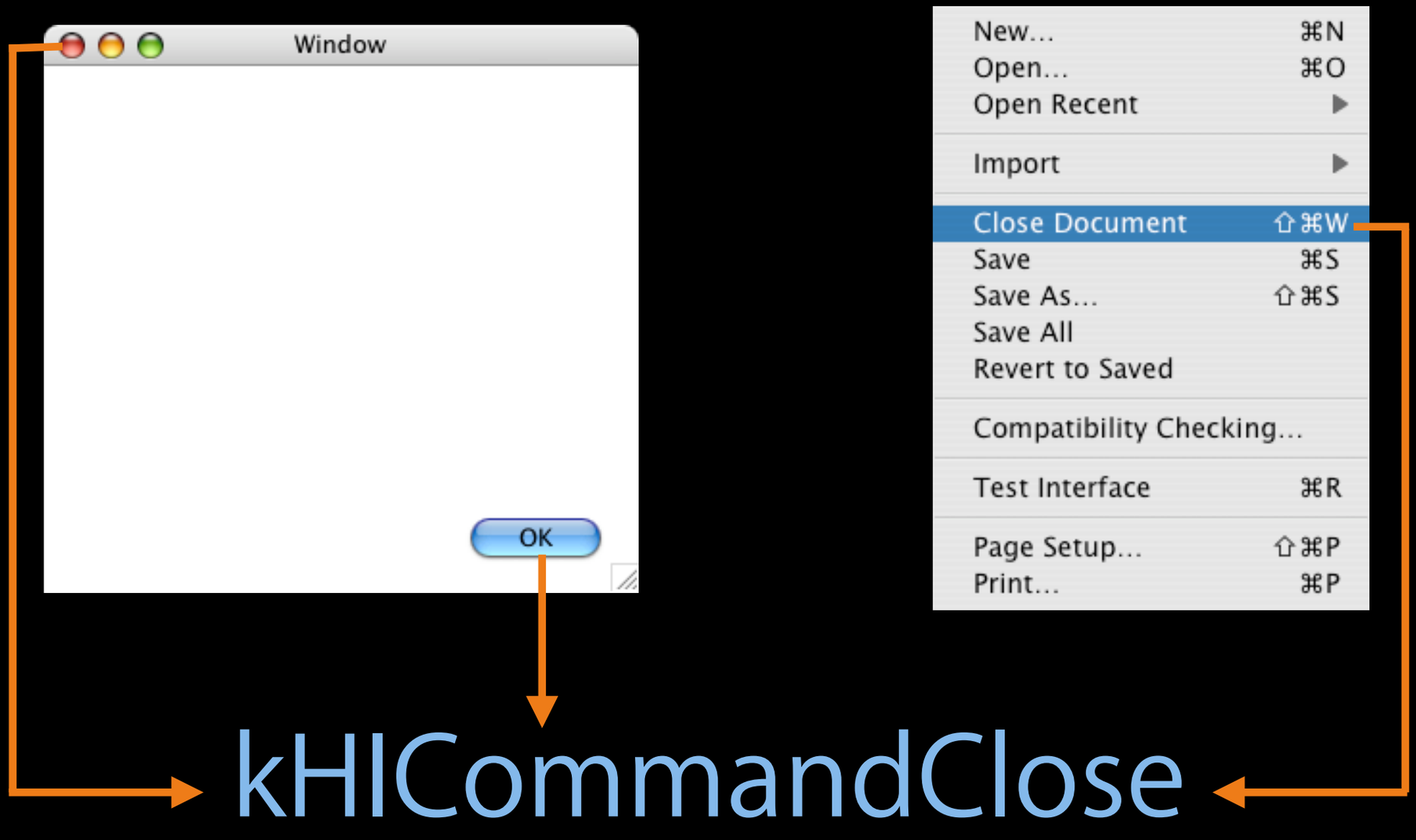
There are a few exceptions

Handling Commands

- What is a command?
  - Application defined user action
  - Source of command is not relevant
- Where do they come from?
  - Menus
  - Controls
- Who handles a command?
  - Depends on the action

Commands: Many-to-one

- The same HICommand can come from multiple sources



Example Command Handler

```
switch ( GetEventClass( inEvent ) )
{
    case kEventClassCommand:
      if ( GetEventKind( inEvent ) == kEventCommandProcess )
      {
          err = GetEventParameter( inEvent, kEventParamDirectObject,
            typeHICommand, NULL, sizeof( HICommand ), NULL,
            &hiCommand );

          switch ( hiCommand.commandID )
          {
            case kHICommandNew:
              ...
            case kHICommandQuit:
              ...
            case kMyCustomCommand:
              ...
          }
      }
    ...
}
```

Example Command Handler

```
switch ( GetEventClass( inEvent ) )
{
    case kEventClassCommand:
      if ( GetEventKind( inEvent ) == kEventCommandProcess )
      {
          err = GetEventParameter( inEvent, kEventParamDirectObject,
            typeHICommand, NULL, sizeof( HICommand ), NULL,
            &hiCommand );

          switch ( hiCommand.commandID )
          {
            case kHICommandNew:
              ...
            case kHICommandQuit:
              ...
            case kMyCustomCommand:
              ...
          }
      }
    ...
}
```

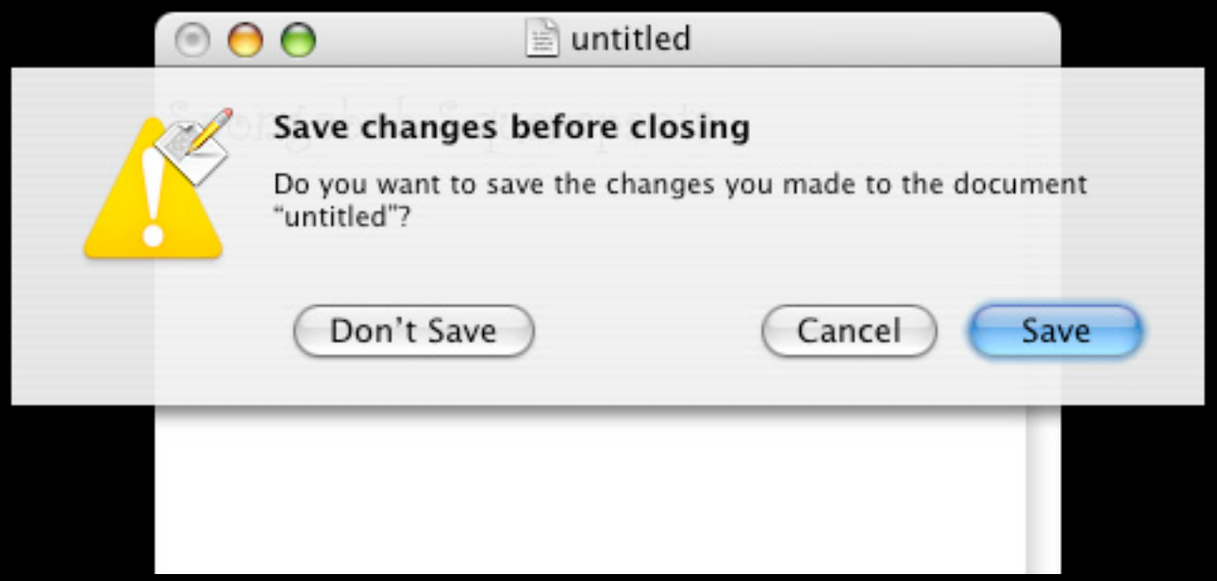
Handling Events and Commands

fullgu

Alerts

- What are they?
  - Inform the user
    - Errors, warnings, get attention
- When to use?

- Standard Alerts
  - Application modal
- Sheet Alerts
  - Window modal



Using Standard Alerts

- Creation

```
OSStatus CreateStandardAlert(
   AlertType          alertType,
   CFStringRef        error,
   CFStringRef        explanation,/* can be NULL */
   const AlertStdCFStringAlertParamRec*
                      param,      /* can be NULL */
   DialogRef*         outAlert );
```

Using Standard Alerts

- Showing the alert

```
OSStatus RunStandardAlert(
   DialogRef          inAlert,
   ModalFilterUPP     filterProc,   /* can be NULL */
   DialogItemIndex*   outItemHit );
```

Using Standard Alerts

- Sample

```
DialogRef               alert;
short                   partHit;

err = CreateStandardAlert(
    kAlertStopAlert,
    CFSTR( "Can’t Save",
    CFSTR( "The file is locked." ),
    NULL, &alert );

err = RunStandardAlert( alert, NULL, &partHit );
```

Using Alert Sheets

- Creation

```
OSStatus CreateStandardSheet(
   AlertType          alertType,
   CFStringRef        error,
   CFStringRef        explanation, // can be NULL
   const AlertStdCFStringAlertParamRec*
                      param,       // can be NULL
   EventTargetRef     notifyTarget, // can be NULL
   DialogRef*         outSheet );
```

Using Alert Sheets

- Showing the sheet

```
OSStatus ShowSheetWindow(
   WindowRef        inSheet,
   WindowRef        inParentWindow );
```

Using Alert Sheets

- Sample

```
DialogRef           sheet;

err = CreateStandardSheet(
   kAlertStopAlert,
   CFSTR( "Can’t Save",
   CFSTR( "The file is locked." ),
   NULL, GetWindowEventTarget( myWindow ),
   &sheet );

err = ShowSheetWindow( GetDialogWindow( sheet ),
   myWindow );
```

What About Files?

- Users need to open or save files as well as choose folders
- Use Navigation Services
  - Does all the work for you
  - Users are used to it
  - When Navigation Services revs, you get the improvements in your application
  - Very customizable

Using Navigation Services

- Choose a file. It's really easy!

```
OSErr NavChooseFile(
  AEDesc*             defaultLocation,/* can be NULL */
  NavReplyRecord*     reply,
  NavDialogOptions*   dialogOptions,  /* can be NULL */
  NavEventUPP         eventProc,      /* can be NULL */
  NavPreviewUPP       previewProc,    /* can be NULL */
  NavObjectFilterUPP  filterProc,     /* can be NULL */
  NavTypeListHandle   typeList,       /* can be NULL */
  void*               callBackUD );   /* can be NULL */
```

Using Navigation Services

- Choose a folder. It's easy too!

```
OSErr NavChooseFolder(
  AEDesc*             defaultLocation,/* can be NULL */
  NavReplyRecord*     reply,
  NavDialogOptions*   dialogOptions,  /* can be NULL */
  NavEventUPP         eventProc,      /* can be NULL */
  NavObjectFilterUPP  filterProc,     /* can be NULL */
  void*               callBackUD );   /* can be NULL */
```

Using Navigation Services

- In Action:

```
NavReplyRecord      reply;
FSRef               folderRef;

err = NavChooseFolder( NULL, &reply, NULL,
   NULL, NULL, NULL );

if ( reply.validRecord == true )
{
   err = AEGetNthPtr( &reply.selection, 1, typeFSRef,
         NULL, NULL, &folderRef,
         sizeof( FSRef ), NULL );
}
```

Using Navigation Services to Select a Folder

Coincidentally, a Navigation demo
by the Guy who changed Nav

Summary

Recap

- Use new HIToolbox to write a modern C/C++ app
- Use Nib files to create great user interface
- Use Carbon Events

Reference Library

Documentation

Handling Carbon Events

ADC Home > Documentation > Carbon > Events & Other Input

Handling Carbon Windows and Controls

ADC Home > Documentation > Carbon > User Experience

Introducing HIView

ADC Home > Documentation > Carbon > User Experience

Reference Library

Documentation (Cont.)

Aqua Human Interface Guidelines

ADC Home > Documentation > Carbon > User Experience

Carbon Porting Guide

ADC Home > Documentation > Carbon > Porting

Reference Library

Sample Code

HIViewTest (and others)

ADC Home > Sample Code > Human Interface Toolbox > Mac OS X
HIToolbox

AppearanceSample (and others)

/Developer/Examples/Carbon in your Developer Tools

Documentation

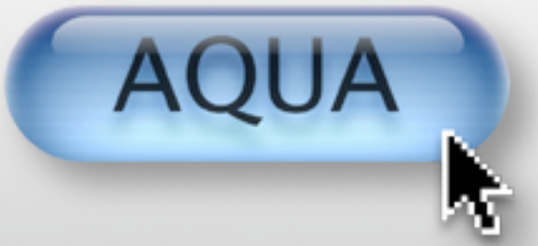
HIToolbox

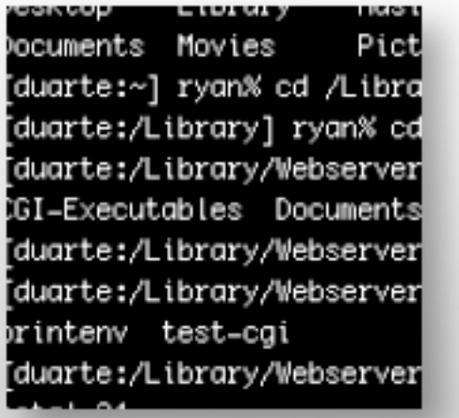
- Header Documentation
  - Header comments, especially for newer work, is very thorough in HIToolbox header files
  - CarbonEvents.h
  - HIView.h
  - Controls.h
  - MacWindows.h
  - /System/Library/Frameworks/Carbon.framework/Frameworks/HIToolbox.framework/Headers

Roadmap

409 Carbon: The New HIToolbox	The Presidio Wed., 9:00am
307 Using Apple Tools for Carbon Development	Russian Hill Thu., 9:00am
416 Carbon: Optimizing Performance	The Presidio Thu., 2:00pm
425 Carbon: HIView in Depth	The Presidio Fri., 3:30pm

Q&A





Xavier Legros

Mac OS X Evangelist *(I Think His Accent Is Fake)*

xavier@apple.com

http://developer.apple.com/wwdc2003/urls.html

WWDC 2003

Worldwide Developers Conference